Design Pattern

Le Pattern Composite

Design Pattern

Un pattern décrit à la fois un problème qui se produit très fréquemment dans l'environnement, et l'architecture de la solution à ce problème de telle façon que l'on puisse utiliser cette solution des milliers de fois sans jamais l'adapter deux fois de la même manière

Principe du Composite Pattern

- **Problème** :

  Etablir des structures arborescentes entre des objets et les traiter uniformément.

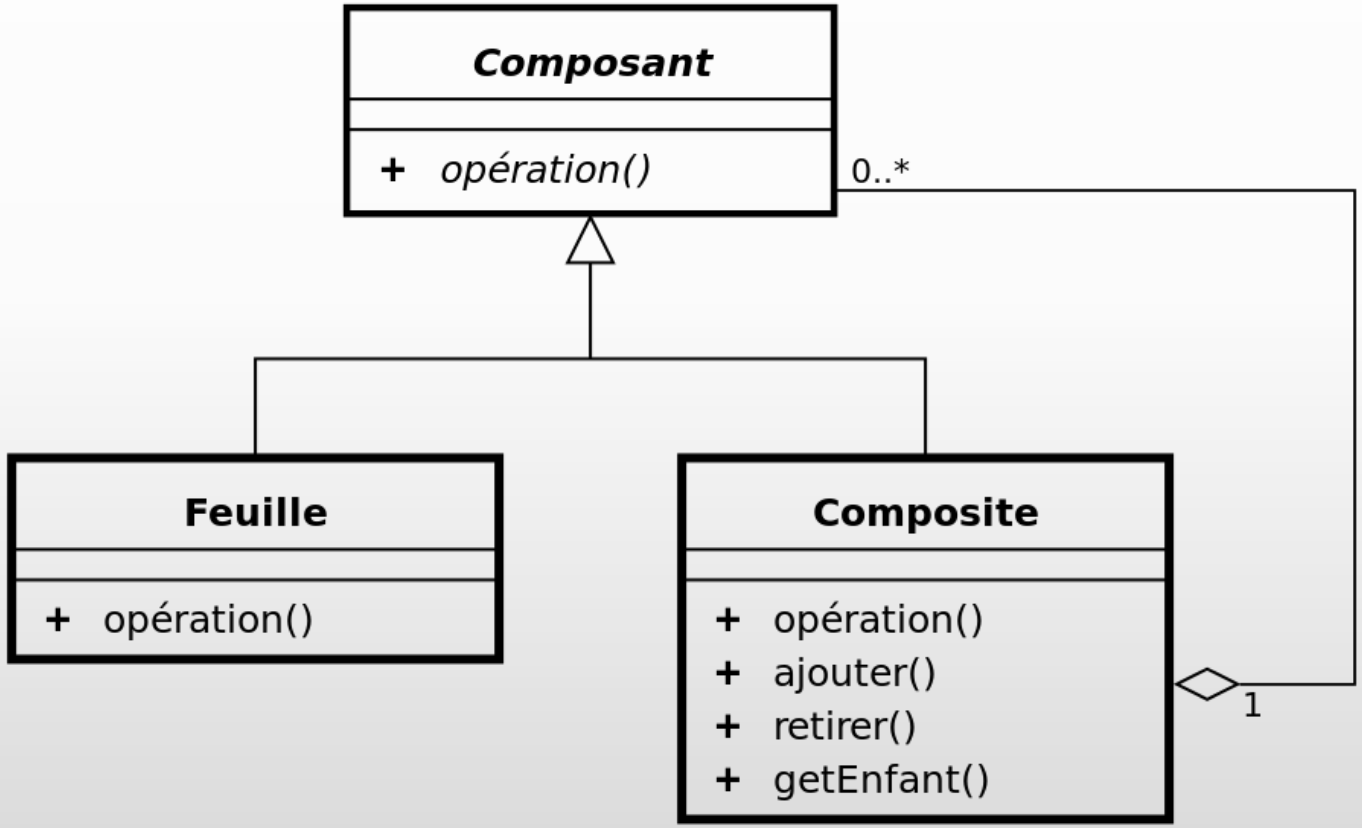
- **Conséquences** :

  Ajout de composants dans la hiérarchie simple.

  L'utilisateur n'a pas à se préoccuper du type d'objet auquel il accède.

  Détermination du type de l'objet plus dure.

Diagramme de classe :



Structure du Composite Pattern :

- **Composant**:

  Interface ou classe abstraite comportant les méthodes communes à tous les objets.

- **Feuille**

  Correspond à un composant n’ayant pas de sous-éléments.

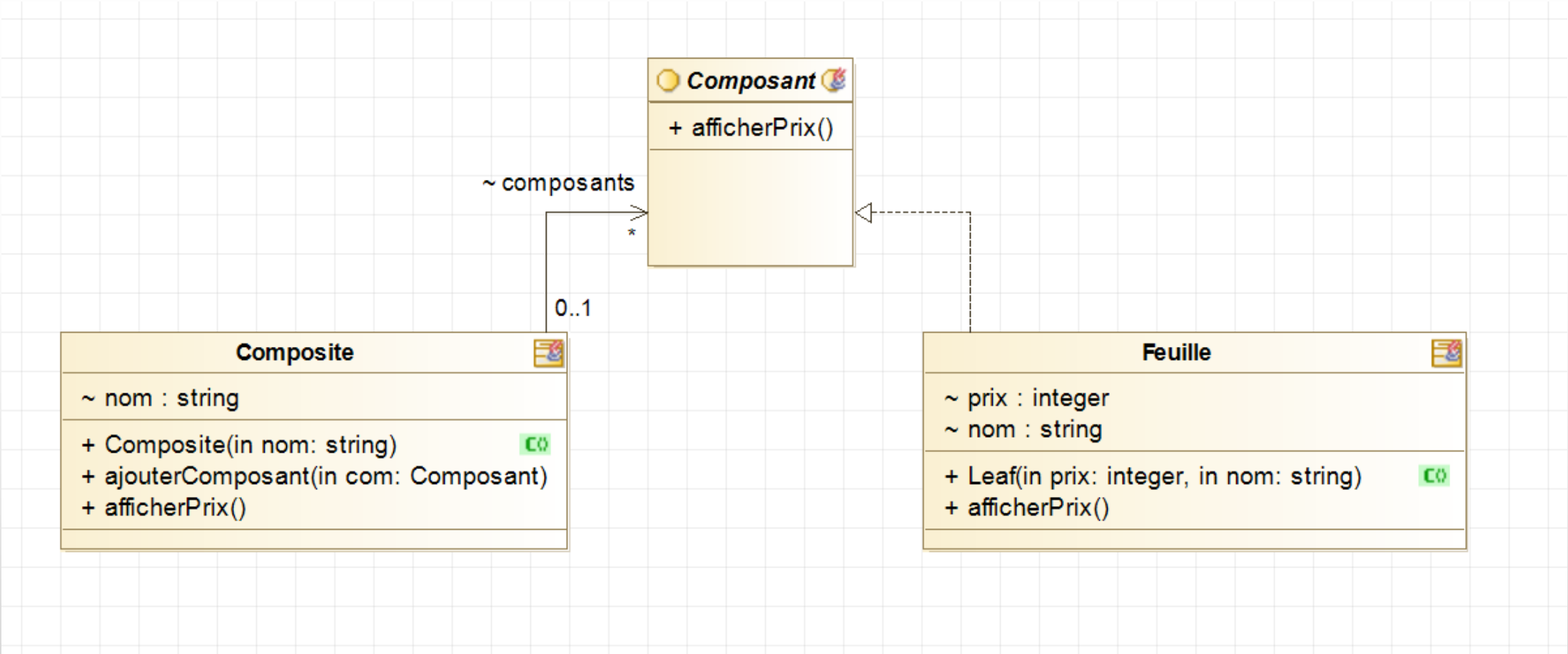
  Suit le comportement défini par défaut.

- **Composite**

  Correspond à un composant qui peut avoir des sous-éléments.

  Suit le comportement défini par défaut.

Exemple : Ordinateur



```java
package fr.unilim.iut.patterncomposite;

import java.util.ArrayList;
import java.util.List;

interface Composant {
    void afficherPrix();
}
class Composite implements Composant {

    String nom;
    List <Composant> composants = new ArrayList<Composant>();


    public Composite(String nom) {
        super();
        this.nom = nom;
    }

    public void ajouterComposant(Composant com){
        composants.add(com);
    }
    public void afficherPrix(){
        System.out.println(nom);
        for(Composant c : composants){
            c.afficherPrix();
        }
    }
}
```

```java
class Leaf implements Composant {

    int prix;
    String nom;

    public Leaf(int prix, String nom){
        super();
        this.prix = prix;
        this.nom = nom;
    }
```

```java
package fr.unilim.iut.patterncomposite;

public class CompositeTest {

    public static void main(String[] args){
        Composant disqueDur = new Leaf(150, "Disque Dur");
        Composant souris = new Leaf(50, "Souris");
        Composant clavier = new Leaf(50, "Clavier");
        Composant processeur = new Leaf(300, "Processeur");
        Composant barretteRam = new Leaf(150, "Barrette RAM");

        Composite carteMere = new Composite("Carte Mere");
        Composite peripheriques = new Composite("Peripheriques");
        Composite boitier = new Composite("Boitier");
        Composite ordinateur = new Composite("Ordinateur");

        carteMere.ajouterComposant(barretteRam);
        carteMere.ajouterComposant(processeur);

        peripheriques.ajouterComposant(souris);
        peripheriques.ajouterComposant(clavier);

        boitier.ajouterComposant(disqueDur);
        boitier.ajouterComposant(carteMere);

        ordinateur.ajouterComposant(boitier);
        ordinateur.ajouterComposant(peripheriques);

        disqueDur.afficherPrix();
        System.out.println(" - - - ");
        boitier.afficherPrix();
        System.out.println(" - - - ");
        ordinateur.afficherPrix();
    }
}
```

Résultats obtenus :

```
Disque Dur : 150
 - - -
Boitier
Disque Dur : 150
Carte Mere
Barrette RAM : 150
Processeur : 300
 - - -
Ordinateur
Boitier
Disque Dur : 150
Carte Mere
Barrette RAM : 150
Processeur : 300
Peripheriques
Souris : 50
Clavier : 50
```

En conclusion :

Le design pattern composite permet de concevoir une structure d’arbre, constitué d’un ou plusieurs objets ayant des fonctionnalités similaires. Le but étant de ne réaliser qu’un seul objet, ils ont ainsi les mêmes opérations, comme dans notre exemple avec + afficherPrix().